The Chinese Postman Problem with load-dependent costs

Ángel Corberán[1], Güneş Erdoğan[2], Gilbert Laporte[3], Isaac Plana[4], José M. Sanchis[5]

[1] Departamento de Estadística e Investigación Operativa, Universitat de València (Spain), angel.corberan@uv.es
[2] School of Management, University of Bath (United Kingdom), g.erdogan@bath.ac.uk
[3] Canada Research Chair in Distribution Management, HEC Montréal (Canada), gilbert.laporte@cirrelt.ca
[4] Departamento de Matemáticas para la Economía y la Empresa, Universitat de València (Spain), isaac.plana@uv.es
[5] Departamento de Matemática Aplicada, Universidad Politécnica de Valencia (Spain), jmsanchis@mat.upv.es

April 25, 2017

Abstract

We introduce an interesting variant of the well-known Chinese Postman Problem (CPP). While in the CPP the cost of traversing an edge is a constant (equal to its length), in the variant we present here the cost of traversing an edge depends on its length and on the weight of the vehicle at the moment it is traversed. This problem is inspired by the perspective of minimizing pollution in transportation, since the amount of pollution emitted by a vehicle not only depends on the travel distance, but also on its load, among other factors. We define the problem, study its computational complexity, provide two mathematical programming formulations and propose two metaheuristics for its solution. Extensive computational experiments reveal the extraordinary difficulty of this problem.

Keywords: Chinese Postman Problem, arc routing problems, pollution-routing.

1 Introduction

In recent years we have witnessed a growing interest in the study of vehicle routing problems that incorporate CO_2 emissions in their objective function. Early contributions include those of McKinnon [25] and of Sbihi and Eglese [31]. The more recent models are rooted in the work of Fagerholt et al. [15], Norstad et al. [26] and Hvattum et al. [18] who solved speed optimization problems in the context of ship routing in order to reduce fuel consumption, and in the paper of Bektaş and Laporte [5] on the "Pollution-Routing Problem", who studied the same issue, this time in the context of vehicle routing. These studies have been followed by several related modeling and algorithmic contributions, namely those of Demir et al.

[11, 12, 13], Koç et al. [20, 21, 22], Kramer et al. [23] and Dabia et al. [10]. For reviews, see Demir et al. [14] and Bektaş et al. [4].

The most common fuel consumption model used in this context is based on the works of Barth et al. [3] and Barth and Boriboonsomsin [2], and to some extent Ross [30]. This model, which is rather involved, comprises three modules, namely engine power, engine speed, and fuel rate. The three main explanatory variables of fuel consumption are the distance traveled, the vehicle weight (including the curb weight and the load carried), and the speed. Several less important factors such as road gradient, acceleration and deceleration, the use of air conditioning, driving style, etc., also come into play, but these are rather difficult to measure. When speed is constant, fuel consumption can be approximated by $A \times \textit{distance} \times \textit{vehicle weight}$, where A is a constant. This is the model that was used by Kara et al. [19].

A problem quite related to the Chinese Postman Problem with load-dependent costs is the Load Dependent Vehicle Routing Problem (LDVRP), studied by Zachariadis et al. [33]. In this problem, the constraint set is exactly the same as that of the VRP, whereas the objective function corresponds to the total product of the distance traveled and the gross weight transported along this distance.

Until now, this line of research has been developed exclusively in the context of node routing. In this paper, we present, as far as we know, the first arc routing problem [9] in which the cost of traversing an edge depends not only on its length, but also on the weight of the vehicle (curb weight plus load) at the moment it is traversed. In particular, the cost of traversing an edge is computed as its length multiplied by the weight of the vehicle. This cost approximates the amount of pollution emitted by a vehicle traveling at constant speed. In Section 2 we define the problem, discuss its characteristics, prove that it is strongly NP-hard, and show that it is polynomially solvable in some special cases. In Section 3 we provide two mathematical programming formulations, one based on a pure arc routing representation, and a second one based on a transformation of the problem into a node routing problem. Some computational results for both formulations are given. In Section 4 two metaheuristics are proposed for the approximate solution of the problem and some computational results on a large set of instances are provided. Conclusions follow in Section 5.

2 The problem

The Chinese Postman Problem with load-dependent costs (CPP-LC) is a very interesting variant of the well-known Chinese Postman Problem [24] that can be defined as follows. Let $G = (V, E)$ be an undirected connected graph, where $V = \{1, \ldots, n\}$ is the vertex set and $E = \{e_1, \ldots, e_m\}$ is the edge set. Vertex 1 represents the depot. Each edge $e \in E$ has a length $d_e \geq 0$ and a demand of $q_e \geq 0$ units of commodity (e.g. kilograms of salt) to be spread on edge e. A vehicle with curb weight W and loaded with $Q = \sum_{e \in E} q_e$ units of commodity starts at the depot, traverses all the edges of the graph servicing the demands and comes back to the depot. The first time an edge $e = (i, j)$ is traversed, it is served, that is, an amount q_e of commodity is downloaded from the vehicle. After being served, an edge e can be traversed in deadheading mode any number of times. Each time the vehicle traverses an edge e (either serving it or deadheading it) it incurs a cost proportional to the edge length d_e multiplied by the current weight of the vehicle:

$$d_e \times \big(W + \text{“load of the vehicle while traversing } e\text{”}\big).$$

While d_e and W are constants, the load is a variable that depends on the amount of commodity downloaded on the edges served before e, and also on edge e itself (while serving it). Let Q_e be the vehicle load at node i just before traversing edge $e = (i, j)$. Then, the cost of traversing e in deadhead is $d_e(W + Q_e)$, while the cost of traversing e while serving it is $d_e(W + Q_e - \frac{q_e}{2})$, because the load of the vehicle upon reaching node j is $Q_e - q_e$, and we assume that the average load of the vehicle while serving the edge e is $Q_e - \frac{q_e}{2}$.

A *CPP-LC tour* is any closed walk, starting and ending at the depot, that traverses all the edges at least once. The cost of a given CPP-LC tour is the sum of the costs associated with the edges it traverses. The CPP-LC consists of finding a CPP-LC tour of minimum cost.

It is interesting to point out that the CPP-LC could be defined by allowing the possibility of deadheading an edge before its service, but this is clearly suboptimal. Also note that the CPP-LC could be defined as a loading problem, where an empty vehicle leaves the depot and loads q_e units of commodity the last time each edge e is traversed. This version is equivalent to the downloading version, and it can be seen that the optimal tour for one version is exactly the reverse of the optimal tour for the other.

To illustrate the features of this problem, consider the CPP-LC instance depicted in Figure 1, where the curb weight of the vehicle is $W = 0$. The two numbers next to an

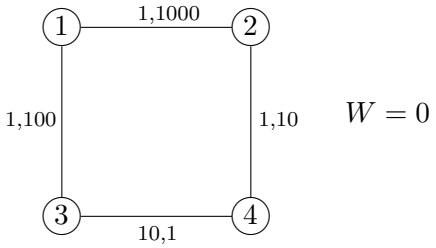

Figure 1: CPP-LC instance

edge e correspond to d_e and q_e. Consider the (Eulerian) CPP-LC tour defined by the walk $(1,2),(2,4),(4,3),(3,1)$ (here (i,j) means that the corresponding edge is traversed and served from i to j). The total distance traveled is $1+1+10+1 = 13$, while its cost is 1772, obtained by adding the cost of all the edge traversals:

$(1,2)$	(vehicle load: 1111)	cost: $1 \times \big(0 + 1111 - \frac{1000}{2}\big)$	$= 611$
$(2,4)$	(vehicle load: 111)	cost: $1 \times \big(0 + 111 - \frac{10}{2}\big)$	$= 106$
$(4,3)$	(vehicle load: 101)	cost: $10 \times \big(0 + 101 - \frac{1}{2}\big)$	$= 1005$
$(3,1)$	(vehicle load: 100)	cost: $1 \times \big(0 + 100 - \frac{100}{2}\big)$	$= 50$
		Total cost:	1772.

It is easy to see that the ‘reverse’ CPP-LC tour $(1,3),(3,4),(4,2),(2,1)$ has a total length of 13, but a cost of 12671. In any case, neither of the above two tours is optimal. The optimal CPP-LC tour for this instance is $(1,2),[2,1],(1,3),[3,1],[1,2],(2,4),(4,3),[3,1]$, where “$[2,1]$” means that the corresponding edge is deadheaded. Although the total length

of this tour is larger (17 versus 13), it has a lower cost of 816:

$(1,2)$	(vehicle load: 1111)	cost: $1 \times \left(0 + 1111 - \frac{1000}{2}\right)$	$= 611$
$[2,1]$	(vehicle load: 111)	cost: $1 \times \left(0 + 111\right)$	$= 111$
$(1,3)$	(vehicle load: 111)	cost: $1 \times \left(0 + 111 - \frac{100}{2}\right)$	$= 61$
$[3,1]$	(vehicle load: 11)	cost: $1 \times \left(0 + 11\right)$	$= 11$
$[1,2]$	(vehicle load: 11)	cost: $1 \times \left(0 + 11\right)$	$= 11$
$(2,4)$	(vehicle load: 11)	cost: $1 \times \left(0 + 11 - \frac{10}{2}\right)$	$= 6$
$(4,3)$	(vehicle load: 1)	cost: $10 \times \left(0 + 1 - \frac{1}{2}\right)$	$= 5$
$[3,1]$	(vehicle load: 0)	cost: $1 \times \left(0 + 0\right)$	$= 0$
		Total cost:	816.

We now discuss the impact of the curb weight W. Consider the same instance as in Figure 1, except for the curb weight W, which is now a positive number. In this case, the cost of the previous CPP-LC tour $(1,2), [2,1], (1,3), [3,1], [1,2], (2,4), (4,3), [3,1]$ increases by W times the total distance traveled, up to $816 + 17W$. On the other hand, the cost of the Eulerian CPP-LC tour $(1,2), (2,4), (4,3), (3,1)$ increases by W times the total distance traveled, up to $1772 + 13W$. Note that when the value for W is sufficiently large ($W > 239$), this Eulerian tour becomes optimal, and it can be seen that, in general, CPP-LC instances with very large values for the curb weight W (compared with the demand values) have the same optimal tour as the classical CPP. Therefore, both problems are equivalent on such instances. Moreover, the CPP-LC with $W > 0$ and $q_e = 0$ for all edges $e \in E$ is equivalent to the CPP.

Note that there are some significant differences between the CPP-LC and the classical CPP, namely:

1. The CPP-LC tours cannot be expressed simply as an (augmented) Eulerian graph but as a sequence of vertices and edges.
2. The minimum-cost augmented graph may be very different from the optimal CPP-LC tour.
3. The optimal CPP-LC tour on an Eulerian graph does not necessarily have to be an Eulerian tour of the graph.
4. An edge can be traversed more than twice (deadheaded more than once) in an optimal CPP-LC tour.

The above differences with respect to the classical CPP are shared by the Cumulative Chinese Postman Problem ([27], [28]). Given an undirected connected graph, the Cumulative Chinese Postman Problem consists of finding a path starting at the depot that services all the edges exactly once such that the total latency is minimized. An edge is served when it is visited for the first time, and its latency is the sum of the latencies of the edges previously traversed by the path, plus the cost of this edge. However, there seems to be no other relevant similarities between the Cumulative CPP and the CPP-LC.

2.1 The CPP-LC is strongly NP-hard

In order to prove that the CPP-LC is a strongly NP-hard problem, we need to recall the definition of the Minimum Latency Problem (MLP). Given a graph with n vertices $1, \ldots, n$ and a distance between any pair of vertices, the MLP consists of finding a tour T starting at 1 and visiting all vertices, in such a way the sum of the arrival times $d_T(1, i)$ (also called latencies) is minimum, where the arrival time is defined as the traveled distance from 1 to i in tour T. The MLP exists in both its closed [7] and its open [1] versions. Sitters [32] proved that the MLP on trees (the problem where the metric is given by an edge-weighted tree) is strongly NP-hard. Although the proof is for the closed version, it can be seen that the open version of the MLP is also strongly NP-hard (a closed MLP instance can be transformed into an open one by just adding an artificial vertex joined to the depot with an edge having a sufficiently large travel time).

An MLP tour $T = \{v_1, v_2, \ldots, v_n\}$ is called *dominated* if there are two vertices v_i and v_{i+1} such that the shortest path from v_i to v_{i+1} contains a vertex v_j with $j > i + 1$. Note that a dominated MLP tour cannot be optimal.

Theorem 1 *The CPP-LC defined on a weighted tree is strongly NP-hard.*

Proof: Consider an open MLP instance on a tree $G = (V, E)$, where $V = \{1 \ldots, n\}$ and E contains $n - 1$ edges not forming cycles, and let d_e be a cost or distance associated with each edge $e \in E$. Let be $D = \sum_{e \in E} d_e$ and, given two vertices $v, w \in V$, let $d(v, w)$ denote the length of the unique path in G joining v and w. We build the CPP-LC instance on the same tree $G = (V, E)$ with the same edge lengths d_e, loads $q_e = 1$ for all $e \in E$, and $W = 0$. It suffices to prove that, for each non-dominated MLP tour, there exists a CPP-LC tour with the same cost (except for a constant) and vice versa.

Consider an MLP tour $T = \{1, v_2, \ldots, v_n\}$, where $\{v_2, \ldots, v_n\}$ is an ordering of the set $\{2, \ldots, n\}$ and assume T is not dominated, that is, in the (unique) path from vertex 1 to v_i there is not any vertex v_j with $j < i$. This tour has the following latency cost:

$$
\begin{aligned}
& d(1, v_2) + \\
& + d(1, v_2) + d(v_2, v_3) + \\
& + d(1, v_2) + d(v_2, v_3) + d(v_3, v_4) + \\
& + \cdots + \\
& + d(1, v_2) + d(v_2, v_3) + d(v_3, v_4) + \cdots + d(v_{n-1}, v_n) = \\
& = (n-1)d(1, v_2) + (n-2)d(v_2, v_3) + (n-3)d(v_3, v_4) + \cdots + d(v_{n-1}, v_n).
\end{aligned}
$$

This MLP tour T defines a path in the graph: $1, v_2, \ldots, v_3, \ldots, v_{n-1}, \ldots, v_n$. Let us denote by w_i the vertex visited just before vertex v_i, $i \geq 3$:

$$1, v_2, w_3, v_3, \ldots, w_4, v_4, \ldots, w_{n-1}, v_{n-1}, \ldots, w_n, v_n.$$

Given that T is a non-dominated tour, each edge (w_i, v_i) is traversed for the first time and, hence, the following is a CPP-LC tour:

$$(1, v_2), [v_2, w_3], (w_3, v_3), [v_3, \ldots, w_4], (w_4, v_4), \ldots, (w_{n-1}, v_{n-1}), [v_{n-1}, \ldots, w_n], (w_n, v_n), [v_n, 1],$$

where $[v_{i-1}, \ldots, w_i]$ represents the traversal in deadheading of the edges in the tree from v_{i-1} to w_i. Given that the initial load is $Q = n - 1$ and $W = 0$, the cost of this CPP-LC solution

is

$$
\begin{aligned}
&d(1,v_2)(n-1-\tfrac{1}{2})+d(v_2,w_3)(n-2)+\\
&+d(w_3,v_3)(n-2-\tfrac{1}{2})+d(v_3,w_4)(n-3)+\\
&+d(w_4,v_4)(n-3-\tfrac{1}{2})+d(v_4,w_5)(n-4)+\\
&+\cdots+\\
&+d(w_{n-1},v_{n-1})(2-\tfrac{1}{2})+d(v_{n-1},w_n)\cdot 1+\\
&+d(w_n,v_n)\cdot 1+d(v_n,1)\cdot 0=\\
&=(n-1)d(1,v_2)+(n-2)d(v_2,v_3)+(n-3)d(v_3,v_4)+\cdots+d(v_{n-1},v_n)-\tfrac{1}{2}D.
\end{aligned}
$$

Conversely, consider a CPP-LC tour

$$(1,v_2),[v_2,w_3],(w_3,v_3),[v_3,\ldots,w_4],(w_4,v_4),\ldots,(w_{n-1},v_{n-1}),[v_{n-1},\ldots,w_n],(w_n,v_n),[v_n,1].$$

Since each edge (w_i,v_i) is serviced the first time it is traversed, it can be seen that each v_i is different from all the vertices $1,v_2,\ldots,v_{i-1}$ (for example, if $v_i=v_{i-m}$ for some $m\geq 1$, the fact that the graph is a tree implies that the edge (v_{i-m},w_i) should have been traversed before $(w_i,v_{i-m})=(w_i,v_i)$). Hence, $T=\{1,v_2,\ldots,v_n\}$ is a non-dominated MLP tour.

Proceeding as above, it can be seen that the cost of the CPP-LC tour is $(n-1)d(1,v_2)+(n-2)d(v_2,v_3)+(n-3)d(v_3,v_4)+\cdots+d(v_{n-1},v_n)-\frac{1}{2}D$ and the cost of the corresponding MLP tour is $(n-1)d(1,v_2)+(n-2)d(v_2,v_3)+(n-3)d(v_3,v_4)+\cdots+d(v_{n-1},v_n)$. ♦

Hence the CPP-LC is strongly NP-hard, although, as will be seen in Section 2.2, it can be solved in polynomial time in some special cases.

2.2 Some polynomially solvable cases

We will now show that the CPP-LC is solvable in polynomial time if it is defined on a star graph. Furthermore, we will consider the special case of the CPP-LC when all the demands are proportional to the distances.

Theorem 2 *The CPP-LC on a star graph is solvable in polynomial time.*

Proof: Let G be a star graph with m edges incident with the depot. We will prove that the optimal solution can be obtained by serving the edges e_i in non-increasing order of q_{e_i}/d_{e_i}. Consider a CPP-LC tour serving the edges in order $e_1,\ldots,e_m$ (renumbering if necessary), so that the tour will traverse each edge twice consecutively. This CPP-LC tour has a cost equal to

$$
\sum_{j=1}^{m}\left(d_{e_j}\big(W+Q-\sum_{i=1}^{j-1}q_{e_i}-\frac{q_{e_j}}{2}\big)+d_{e_j}\big(W+Q-\sum_{i=1}^{j-1}q_{e_i}-q_{e_j}\big)\right)=
$$

$$
=\sum_{j=1}^{m}d_{e_j}\big(2W+2Q-2\sum_{i=1}^{j-1}q_{e_i}-\frac{3}{2}q_{e_j}\big)=2\sum_{j=1}^{m}d_{e_j}(W+Q)+\sum_{j=1}^{m}d_{e_j}\frac{q_{e_j}}{2}-2\sum_{j=1}^{m}d_{e_j}\sum_{i=1}^{j}q_{e_i}.
$$

Note that the first two terms of this expression do not depend on the order in which the edges are served. Now consider an instance of the problem of scheduling jobs with release dates on a single machine with m jobs, release dates $r_i=0$, processing times $p_i=q_{e_i}$, and

weights $w_i = d_{e_i}$. The objective function of this problem is

$$\sum_{j=1}^{m} w_j \sum_{i=1}^{j} p_i = \sum_{j=1}^{m} d_{e_j} \sum_{i=1}^{j} q_{e_i}$$

and is minimized when the jobs are sequenced in non-decreasing order of p_i/w_i [6]. Hence, an optimal CPP-LC tour is obtained when the edges are sequenced in non-increasing order of q_{e_i}/d_{e_i}. ♦

In what follows, we will focus on a special case of the CPP-LC, which we call the *Proportional* CPP-LC (PCPP-LC), where demands q_e are proportional to the lengths d_e, that is, $q_e = \beta\, d_e,\ \forall\, e \in E$, for some $\beta > 0$. This corresponds to some practical situations in which the amount of commodity downloaded (for example, salt or water) is proportional to the length of the street.

To illustrate the features of this version of the problem, consider the PCPP-LC instance depicted in Figure 2 with $\beta = 10$. It can be seen that the two possible Eulerian tours are optimal solutions with cost 62720.

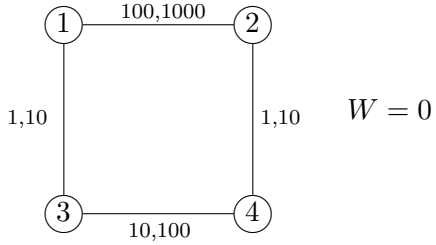

Figure 2: PCPP-LC instance

Proposition 1 *Let $G = (V, E)$ be a PCPP-LC instance. If G is an Eulerian graph, all the Eulerian PCPP-LC tours have the same cost.*

Proof: It is easy to see that, without loss of generality, we can assume that $\beta = 1$. Consider an Eulerian tour that traverses the edges (renumbering, if necessary) in the following order: $e_1, e_2, e_3, e_4, \cdots, e_m$. This PCPP-LC tour has a cost equal to

$$\sum_{j=1}^{m} q_{e_j}\big(W + Q - \sum_{i=1}^{j-1} q_{e_i} - \frac{q_{e_j}}{2}\big) = W\sum_j q_{e_j} + Q\sum_j q_{e_j} - \sum_{r\neq s} q_{e_r} q_{e_s} - \sum_j \frac{q_{e_j}^2}{2} =$$

$$= W\sum_j q_{e_j} + \big(\sum_j q_{e_j}\big)^2 - \sum_{r\neq s} q_{e_r} q_{e_s} - \sum_j \frac{q_{e_j}^2}{2} \overset{(*)}{=} QW + \sum_j q_{e_j}^2 + 2\sum_{r\neq s} q_{e_r} q_{e_s} - \sum_{r\neq s} q_{e_r} q_{e_s} - \sum_j \frac{q_{e_j}^2}{2} =$$

$$= QW + \frac{1}{2}\Big(\sum_j q_{e_j}^2 + 2\sum_{r\neq s} q_{e_r} q_{e_s}\Big) \overset{(*)}{=} QW + \frac{1}{2}\big(\sum_j q_{e_j}\big)^2 = QW + \frac{1}{2}Q^2,$$

where the equalities labeled as $(*)$ follow from the equality $\big(\sum_i a_i\big)^2 = \sum_i a_i^2 + 2\sum_{i\neq j} a_i a_j$. Since the total cost does not depend on the ordering of the edges, all the Eulerian PCPP-LC tours have the same cost $QW + \frac{1}{2}Q^2$. ♦

The cost of a CPP-LC tour can be seen as the sum of its *serving cost* (the sum of the costs of the edges traversed when they are served) and its *deadheading cost* (the sum of the costs of the edges traversed in deadhead).

Proposition 2 *Let $G = (V, E)$ be a PCPP-LC instance. The serving cost of any PCPP-LC tour is $QW + \frac{1}{2}Q^2$.*

Proof: For the sake of simplicity, suppose that the tour contains only one deadheaded edge (edge e, for example) and that, as in Proposition 1, it traverses the edges (renumbering, if necessary) in the following order: e_1, e_2, e_3, e, e_4, $\cdots$, e_m. This PCPP-LC tour has a cost equal to

$$q_{e_1}\big(W + Q - \tfrac{q_{e_1}}{2}\big) + q_{e_2}\big(W + Q - q_{e_1} - \tfrac{q_{e_2}}{2}\big) + q_{e_3}\big(W + Q - q_{e_1} - q_{e_2} - \tfrac{q_{e_3}}{2}\big) +$$
$$+ q_{e}\big(W + Q - q_{e_1} - q_{e_2} - q_{e_3}\big) + q_{e_4}\big(W + Q - q_{e_1} - q_{e_2} - q_{e_3} - \tfrac{q_{e_4}}{2}\big) + \cdots +$$
$$+ q_{e_m}\big(W + Q - q_{e_2} - q_{e_3} - \cdots - q_{e_{m-1}} - \tfrac{q_{e_m}}{2}\big) = QW + \tfrac{1}{2}Q^2 + q_e\big(W + Q - q_{e_1} - q_{e_2} - q_{e_3}\big),$$

which is $QW + \frac{1}{2}Q^2$, plus the cost of deadheading edge e. This argument can be extended to any number of edges traversed in deadheading, and hence the result. ♦

Solving the PCPP-LC is therefore equivalent to minimizing the deadheading cost, because the serving cost of all the tours is a constant. Hence, the PCPP-LC seems to be easier than the general CPP-LC. In fact, we will see that the PCPP-LC for Eulerian graphs and for weighted trees is solvable in polynomial time.

For Eulerian graphs, all the Eulerian PCPP-LC tours have a deadheading cost equal to zero and we then have the following theorem.

Theorem 3 *If $G = (V, E)$ is a PCPP-LC instance and G is an Eulerian graph then any Eulerian PCPP-LC tour is optimal. Hence, the PCPP-LC for Eulerian graphs is solvable in polynomial time.*

For weighted trees, the following proposition proves that all the PCPP-LC tours deadheading each edge exactly once have the minimum deadheading cost.

Proposition 3 *Let $G = (V, E)$ be a PCPP-LC instance. If G is a tree, all the PCPP-LC tours traversing each edge exactly twice (once servicing it and another time in deadheading) are optimal.*

Proof: Again, assume that $\beta = 1$. From Proposition 2, all the PCPP-LC tours on G have the same serving cost and we can focus on minimizing their deadheading cost.

The deadheading cost of traversing an edge $a \in E$ is the product of its length $d_a = q_a$ by the load in the vehicle, which is the sum of the demands q_b of the edges b that have not yet been served. Hence, the total deadheading cost of a PCPP-LC tour can be expressed as the sum of products $q_a q_b$ for some pairs of edges $a, b \in E$. Each term $q_a q_b$ corresponds to deadheading a while b has not yet been served, or vice versa.

Let $a, b \in E$ be a pair of edges. We will say that a and b are compatible if the (unique) path from the depot to edge b contains edge a, or vice versa. This is the case, for example, for pairs a and d, a and f or e and g in the tree depicted in Figure 3. When neither a is on the path from the depot to b, nor b is on the path from the depot to a, we say that edges a and b are incompatible, as for pairs d and f or a and b in Figure 3.

We will first prove that all the PCPP-LC tours traversing each edge exactly twice (once servicing it and once deadheading) have the same deadheading cost. Let T be any of these tours. Since this tour deadheads each edge exactly once, each term $q_a q_b$ may appear up to twice in the deadheading cost of T: once when deadheading a, and possibly once more when deadheading b.

Consider first a pair $a, b \in E$ of compatible edges. Assume, without loss of generality, that the path from the depot to edge b contains edge a. Then, T serves a before b but it deadheads b before a. Therefore, when b is deadheaded, a has already been served, and vice versa, and the term $q_a q_b$ does not appear in the deadheading cost of T.

Consider now a pair $a, b \in E$ of incompatible edges. Note that either T traverses a twice before traversing b (twice), or vice versa. Assume, without loss of generality, that T serves and deadheads a before b. When T deadheads a, edge b has not yet been served, and when it deadheads b, edge a has been already served. Therefore, the term $q_a q_b$ appears exactly once in the deadheading cost of T for each pair $a, b \in E$ of incompatible edges. Therefore, the total deadheading cost of T is

$$\sum_{a,b \text{ incompatible}} q_a q_b, \tag{1}$$

which is the same value for all the PCPP-LC tours traversing each edge exactly twice.

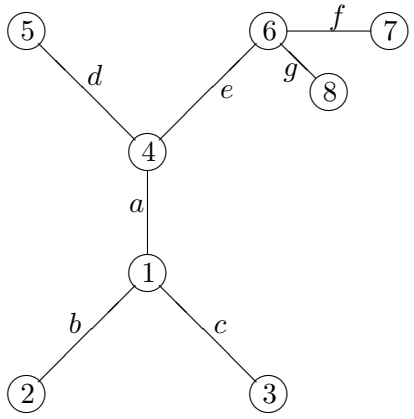

Figure 3: A PCPP-LC instance on a weighted tree

Finally, let T be a PCPP-LC tour on G traversing some edges more than twice. Consider a pair $a, b \in E$ of incompatible edges. Given that G is a tree, either T traverses a at least twice before traversing b (at least twice), or vice versa. Assume, without loss of generality, that T serves and deadheads for the first time a before traversing b. Then, when T deadheads a for the first time, edge b has not been served yet, and the term $q_a q_b$ appears at least once in the deadheading cost of T. Since this is true for each pair $a, b \in E$ of incompatible edges, the total deadheading cost of T is at least equal to the constant in (1). Therefore, all the

PCPP-LC tours traversing each edge exactly twice are optimal. ♦

As a consequence of Propositions 2 and 3, we obtain the following result:

Theorem 4 *The PCPP-LC defined on a weighted tree is solvable in polynomial time.*

3 Two mathematical programming formulations

In this section we present two formulations for the CPP-LC. The first one is an arc routing formulation which tries to exploit the problem characteristics. The second (node routing) formulation is based on a transformation of the original graph into a graph in which the edges are separated, and their endnodes are replicated.

3.1 An arc-routing formulation

Given that the cost of traversing the edges depends on the order in which they are traversed, we consider $K = |E|$ periods, denoted by $k = 1, \ldots, K$. Each period consists of an edge that is serviced at the beginning of the period and all the deadheadings (if any) needed to reach the edge serviced at the beginning of the following period. Let us define the following $4|E|^2 + |E|$ variables:

(Route variables) Four binary variables for each edge (i, j) and for each period $k = 1, \ldots, K$:

- $y_{ij}^k = 1$ if edge $e = (i, j)$ is served (and traversed) from i to j in period k, 0 otherwise,
- $x_{ij}^k = 1$ if edge $e = (i, j)$ is traversed in deadheading in period k, 0 otherwise.

(Load variables) A continuous variable $f_k > 0$ representing the load in the vehicle at the beginning of period k, $k = 2, \ldots, K$. We can assume that $f_1 = Q$ and $f_{K+1} = 0$.

$$1 \xrightarrow[f_1]{y_{12}^1} 2 \xrightarrow[f_2]{x_{21}^1} 1 \xrightarrow[f_2]{y_{13}^2} 3 \xrightarrow[f_3]{x_{31}^2} 1 \xrightarrow[f_3]{x_{12}^2} 2 \xrightarrow[f_3]{y_{24}^3} 4 \xrightarrow[f_4]{y_{43}^4} 3 \xrightarrow[f_5]{x_{31}^4} 1$$

Figure 4: Variables of the optimal tour of the instance in Figure 1

Figure 4 shows the variables associated with the optimal tour for the CPP-LC instance shown in Figure 1 (the edges traversed in service are represented in solid lines while the edges traversed in deadheading are plotted in dashed lines).

Some lower and upper bounds for variables f_k can be computed. For example, for an instance with six edges with demands $q_e = 1, 2, 3, 4, 5$, and 6, we have the following bounds:

$f_1 = 21, \quad 21 - 6 \leq f_2 \leq 21 - 1, \quad 21 - 6 - 5 \leq f_3 \leq 21 - 1 - 2, \quad$ etc.

In fact, if all demands are equal to 1 for example, the f_k values are determined: $f_1 = 6, f_2 = 5, f_3 = 4, f_4 = 3, f_5 = 2, f_6 = 1, f_7 = 0$. In general, if we order the demands in non-decreasing order $q_{e_1} \leq q_{e_2} \leq \cdots \leq q_{e_K}$, we define $L_1 = U_1 = Q$ and, for each $k = 1, \ldots, K$, $L_{k+1} = L_k - q_{e_{K-k+1}}$ and $U_{k+1} = U_k - q_{e_k}$. Then $L_k \leq f_k \leq U_k$ for each $k = 2, \ldots, K$.

In the formulation below we use the following notation. Given a vertex $i \in V$, $x^k(\delta^+(i)) = \sum_{(i,j)\in E} x^k_{ij}$ and $x^k(\delta^-(i)) = \sum_{(i,j)\in E} x^k_{ji}$. The same notation is used for variables y^k_{ij}. The CPP-LC can then be formulated as follows:

$$\text{Minimize} \qquad \sum_{k=1}^{K} \sum_{e=(i,j)\in E} d_e\Big(\big(W + f_k - \frac{q_e}{2}\big)(y^k_{ij} + y^k_{ji}) + \big(W + f_{k+1}\big)(x^k_{ij} + x^k_{ji})\Big)$$

subject to

$$\sum_{k=1}^{K}(y^k_{ij} + y^k_{ji}) = 1, \qquad \forall e \in E \tag{2}$$

$$\sum_{(i,j)\in E} (y^k_{ij} + y^k_{ji}) = 1, \qquad \forall k = 1, \ldots, K \tag{3}$$

$$f_{k+1} = f_k - \sum_{e=(i,j)\in E} q_e \ (y^k_{ij} + y^k_{ji}), \qquad \forall\ k = 1, \ldots, K-1 \tag{4}$$

$$f_1 = Q, \quad f_{K+1} = 0, \tag{5}$$

$$y^1(\delta^-(i)) + x^1(\delta^-(i)) = y^2(\delta^+(i)) + x^1(\delta^+(i)), \qquad \forall i \in V \setminus \{1\} \tag{6}$$

$$y^k(\delta^-(i)) + x^k(\delta^-(i)) = y^{k+1}(\delta^+(i)) + x^k(\delta^+(i)), \qquad \forall i \in V, \ \ \forall\ k = 2, \ldots, K-1 \tag{7}$$

$$y^K(\delta^-(i)) + x^K(\delta^-(i)) = x^K(\delta^+(i)), \qquad \forall i \in V \setminus \{1\} \tag{8}$$

$$y^K(\delta^-(1)) + x^K(\delta^-(1)) = y^1(\delta^+(1)) = 1, \tag{9}$$

$$x^k_{ij} \leq \sum_{l=1}^{k-1}(y^l_{ij} + y^l_{ji}) + y^k_{ji}, \qquad \forall (i,j) \in E, \ \ \forall\ k = 1, \ldots, K \tag{10}$$

$$x^k_{ij}, x^k_{ji}, y^k_{ij}, y^k_{ji} \in \{0, 1\}, \qquad \forall (i,j) \in E, \ \forall\ k = 2, \ldots, K \tag{11}$$

$$L_k \leq f_k \leq U_k, \qquad \forall\ k = 2, \ldots, K. \tag{12}$$

The objective function is the sum of the costs of the edges served, $d_e\big(W + f_k - \frac{q_e}{2}\big)(y^k_{ij} + y^k_{ji})$, and the deadheaded edges, $d_e\big(W + f_{k+1}\big)(x^k_{ij} + x^k_{ji})$. Inequalities (2) and (3) guarantee that each edge is served in one period and that only one edge is served in each period. Constraints (4) ensure that when an edge e is served, the vehicle load is decreased by q_e units. Symmetry conditions on the vertices are imposed by equations (6) to (9). Constraints (10) state that an edge can be deadheaded only if it has been served in a previous period, or in the same period but traversed in the opposite direction. These constraints are not needed to find the optimal CPP-LC tour. Their removal would allow feasible solutions in which some edges could be deadheaded before being served, although, as it has been said, these solutions would never

be optimal. Nevertheless these constraints have proved to be useful from a computational point of view.

The objective function is non-linear because of the terms $f_k\,(y^k_{ij}+y^k_{ji})$ and $f_{k+1}\,(x^k_{ij}+x^k_{ji})$, which are the product of a binary and a continuous variables. By defining the following variables:

- $t^k_e = f_k(y^k_{ij} + y^k_{ji}), \qquad \forall e=(i,j)\in E, \;\; \forall\; k = 1,\ldots,K,$
- $z^k_e = f_{k+1}(x^k_{ij} + x^k_{ji}), \qquad \forall e=(i,j)\in E, \;\; \forall\; k = 1,\ldots,K-1,$

and by adding the constraints

$$\left.\begin{array}{l} t^k_e \geq L_k\,(y^k_{ij} + y^k_{ji}), \\ t^k_e \leq U_k\,(y^k_{ij} + y^k_{ji}), \\ t^k_e \leq f_k + L_k\,(y^k_{ij} + y^k_{ji} - 1), \\ t^k_e \geq f_k + U_k\,(y^k_{ij} + y^k_{ji} - 1), \end{array}\right\} \quad \forall e=(i,j)\in E, \;\; \forall\; k = 1,\ldots,K \tag{13}$$

and

$$\left.\begin{array}{l} z^k_e \geq L_{k+1}(x^k_{ij} + x^k_{ji}), \\ z^k_e \leq U_{k+1}(x^k_{ij} + x^k_{ji}), \\ z^k_e \leq f_{k+1} + L_{k+1}(x^k_{ij} + x^k_{ji} - 1), \\ z^k_e \geq f_{k+1} + U_{k+1}(x^k_{ij} + x^k_{ji} - 1), \end{array}\right\} \quad \forall e=(i,j)\in E, \;\; \forall\; k = 1,\ldots,K-1, \tag{14}$$

to the previous formulation, the objective function can be linearized as

$$\text{Minimize} \quad \sum_{k=1}^{K} \sum_{e=(i,j)\in E} d_e\Big(\big(W-\frac{q_e}{2}\big)(y^k_{ij}+y^k_{ji}) + W(x^k_{ij}+x^k_{ji})\Big) + \sum_{k=1}^{K}\sum_{e\in E} d_e t^k_e + \sum_{k=1}^{K-1}\sum_{e\in E} d_e z^k_e.$$

Some variables can be fixed to 0. For instance, all the y^1 variables of the edges that are not incident with the depot can be fixed to 0. Moreover, if we compute the cost s_{1i} of the shortest path from the depot to a vertex i, using costs equal to 1 for all the edges in G, we can fix $y^k_{ij} = 0$ for all $k < s_{1i}$ and $x^k_{ij} = 0$ for all $k < s_{1,i-1}$.

Note also that $\sum\limits_{e\in E} t^k_e = \sum\limits_{e\in E} f_k(y^k_{ij}+y^k_{ji}) = f_k \sum\limits_{e\in E}(y^k_{ij}+y^k_{ji}) = f_k$, and therefore the equations

$$\sum_{e\in E} t^k_e = f_k, \quad \forall\; k = 1,\ldots,K \tag{15}$$

can be used to strengthen the formulation.

We have conducted computational experiments with this formulation. The results obtained are presented in Section 3.3.

3.2 A node-routing formulation

Because the preliminary computational results obtained with the arc-routing formulation were not entirely satisfactory, we decided to try a different approach. The main drawbacks of the arc-routing formulation are the large number of variables and the non-linearity of the objective function, whose linearization implies an important increase both in the number of variables and constraints. In this section, we propose another formulation based on a transformation of the original graph G. Broadly speaking, it consists of separating all the edges of G and replicating their endnodes.

Figure 5 illustrates the transformation, where the left graph is the original graph $G = (V, E)$ and the right one represents the transformed graph $\bar{G} = (\bar{V}, \bar{E})$. For the sake of simplicity, we have labeled the five edges of G in Figure 5 from 1 to 5. Associated with each edge in E are two vertices in $\bar{V}$. For example, given the edge number 3, $(1, 4) \in E$, the vertices 1_3 and 4_3 in $\bar{V}$ are linked with a 'required' edge plotted in bold line. Vertex $0 \in \bar{V}$ represents an artificial vertex which is the starting and ending point of the route. Graph $\bar{G}$ is a complete graph, that is, in addition to the 'required' edges (corresponding to the edges in G), set $\bar{E}$ contains all the edges between pairs of vertices in $\bar{V}$ not corresponding to the same edge in G. Some of these edges are depicted in Figure 5 by dashed lines.

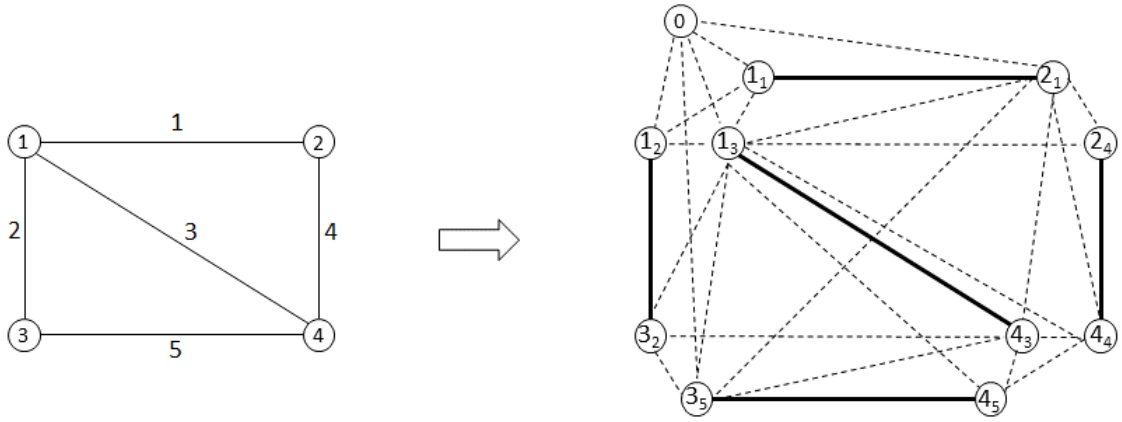

Figure 5: The graph transformation

Let CPP-LC* denote the generalization of the CPP-LC where deadheading an edge before servicing it is allowed. It can be seen that each CPP-LC* tour in G defines a tour in $\bar{G}$. For example, given the CPP-LC* tour $(1, 2), (2, 4), (4, 1), [1, 4], (4, 3), (3, 1)$, we have the tour $(0, 1_1, 2_1, 2_4, 4_4, 4_3, 1_3, 4_5, 3_5, 3_2, 1_2, 0)$, which is a Hamiltonian tour in $\bar{G}$, starting and ending at node 0, traversing each required edge exactly once and alternating required and non-required edges. Conversely, each Hamiltonian tour in $\bar{G}$ with these properties defines a CPP-LC* tour in G. For example, $(0, 1_3, 4_3, 4_5, 3_5, 2_1, 1_1, 1_2, 3_2, 4_4, 2_4, 0)$ defines the CPP-LC* tour $(1, 4), (4, 3), [3, 4], [4, 2], (2, 1), (1, 3), [3, 4], (4, 2), [2, 1]$, in which the edge $(2, 4)$ is deadheaded before being serviced.

Hence, the CPP-LC* can be solved as finding the "Hamiltonian alternating tour" in $\bar{G}$ with minimum cost when we define the appropriate edge costs in $\bar{G}$. This allows to directly solve the CPP-LC because, as already mentioned, deadheading before servicing is suboptimal.

In general, a complete graph $\bar{G} = (\bar{V}, \bar{E})$ is built from the original graph $G = (V, E)$ and the corresponding values for d_e and q_e. The set $\bar{V}$ contains two vertices for each edge

$e = (i, j) \in G$ and a vertex 0 representing the starting and ending point of the tour. The set $\bar{E}$ contains the required edges (those corresponding to edges in G), denoted by R, and the non-required edges (corresponding to deadheading paths in G). Given $i \in \bar{V}$, let $i^* \in V$ be the corresponding vertex in G (with $0^* = 1$). Given $i, v \in \bar{V}$, s_{iv} denotes the cost of a shortest path from i^* to v^* in G. We define two costs $\bar{c}_{ij}$, $\bar{c}_{ji}$ associated with the two traversals of each edge $(i, j) \in \bar{G}$ as follows:

1) For the edges $(i, j) \in R$ ($(i^*, j^*) \in E$):

- $\bar{c}_{ij} = \bar{c}_{ji} = 0$.

2) For the edges $(i, v) \in \bar{E} \setminus R$ such that vertices $i, v \in \bar{V}$ correspond to two different edges $(i^*, j^*), (v^*, w^*) \in E$:

- $\bar{c}_{iv} = s_{iv} + d_{v^*w^*}$,
- $\bar{c}_{vi} = s_{iv} + d_{i^*j^*}$.

3) For the edges incident with vertex 0:

- $\bar{c}_{0i} = s_{1i} + d_{i^*j^*}$,
- $\bar{c}_{i0} = s_{i1}$.

We define the following variables:

(Route variables) For each edge $e = (i, j) \in \bar{E}$, let $x_{ij} = 1$ if $e = (i, j)$ is traversed from i to j, 0 otherwise.

(Load variables) For each edge $e = (i, j) \in \bar{E} \setminus R$, let $f_{ij} \geq 0$ be a continuous variable representing the load in the vehicle when traversing e from i to j. We can assume that $\sum_{i \in V} f_{0i} = Q$ and $f_{i0} = 0$.

Now consider a Hamiltonian tour in $\bar{G}$ starting and ending at node 0 and traversing exactly once each required edge in $\bar{E}$. This tour traverses the edges in $\bar{G}$ alternating non-required and required edges. Its cost is the sum of the costs of pairs of consecutive non-required and required edges $(i, j) \in \bar{E} \setminus R$, $(j, w) \in R$. The distance traveled and the load of the vehicle when traversing (i, j) is s_{ij} and f_{ij}, respectively, while they are $d_{j^*w^*}$ and $f_{ij} - \frac{1}{2} q_{j^*w^*}$ when traversing (j, w). The cost of traversing these two edges is: $s_{ij}\big(W + f_{ij}\big) \; + \; d_{j^*w^*}\big(W + f_{ij} - \frac{1}{2} q_{j^*w^*}\big)$.

Therefore, the cost of the tour corresponding to the curb weight W expressed with the x_{ij} variables is

$$\sum_{(i,j) \in \bar{E} \setminus R} W \, s_{ij} \, x_{ij} \quad + \sum_{(i,j) \in R} W \, d_{i^*j^*} \, x_{ij},$$

and the one corresponding to the load in the vehicle expressed with the f_{ij} variables is

$$\sum_{(i,j) \in \bar{E} \setminus R} \Big(s_{ij} f_{ij} + d_{j^*w^*} \big(f_{ij} - \tfrac{1}{2} \, q_{j^*w^*} \big) \Big) = \sum_{(i,j) \in \bar{E} \setminus R} \bar{c}_{ij} f_{ij} - \sum_{(i,j) \in \bar{E} \setminus R} \tfrac{1}{2} \, d_{j^*w^*} \, q_{j^*w^*} =$$

$$= \sum_{(i,j)\in\bar{E}\setminus R} \bar{c}_{ij} f_{ij} - \frac{1}{2}\sum_{e\in E} d_e\, q_e.$$

The CPP-LC can then be formulated as follows:

$$\text{Minimize} \quad \sum_{(i,j)\in\bar{E}\setminus R} W\, s_{ij}\, x_{ij} \;+\; \sum_{(i,j)\in R} W\, d_{i^*j^*}\, x_{ij} \;+\; \sum_{(i,j)\in\bar{E}\setminus R} \bar{c}_{ij} f_{ij} \;-\; \frac{1}{2}\sum_{e\in E} d_e\, q_e$$

subject to

$$x_{ij} + x_{ji} = 1, \qquad \forall\ (i,j) \in R \tag{16}$$

$$\sum_{j\in\bar{V}} x_{ij} = 1, \qquad \forall i \in \bar{V} \tag{17}$$

$$\sum_{i\in\bar{V}} x_{ij} = 1, \qquad \forall j \in \bar{V} \tag{18}$$

$$\sum_{i\in\bar{V}} f_{iv} = q_{vw}\, x_{vw} + \sum_{j\in\bar{V}} f_{wj}, \qquad \forall (v,w) \in R \tag{19}$$

$$\sum_{i\in\bar{V}} f_{iw} = q_{vw}\, x_{wv} + \sum_{j\in\bar{V}} f_{vj}, \qquad \forall (v,w) \in R \tag{20}$$

$$\sum_{i\in\bar{V}} f_{0i} = Q, \quad \sum_{i\in\bar{V}} f_{i0} = 0, \tag{21}$$

$$q_{vw}\, x_{iv} \le f_{iv} \le \big(Q - q_{ij}\big) x_{iv}, \qquad \forall\ (i,v) \in \bar{E}\setminus R, \text{ with } (i^*,j^*),(v^*,w^*) \in E \tag{22}$$

$$x_{ij} \in \{0,1\}, \qquad \forall (i,j) \in \bar{E} \tag{23}$$

$$0 \le f_{ij} \le Q, \qquad \forall (i,j) \in \bar{E}\setminus R. \tag{24}$$

Constraints (16) ensure each required edge to be traversed exactly once in one of the two possible directions. Equations (17) and (18) guarantee that the tour visits all the vertices. When the vehicle serves a required edge (v,w), the load in the vehicle decreases by its demand q_{vw} (constraints (19) and (20)). Finally, inequalities (22) link the route and load variables.

3.3 Computational results for the two formulations

In order to test the two formulations we have generated the following sets of CPP-LC instances:

- 18 Eulerian instances, whose name starts with E, obtained from three Eulerian graphs, with $|V| = 7, 10, 20$ and $|E| = 12, 18, 32$, respectively. For each graph, three different values of W have been considered, $W = 0, \frac{Q}{2}, 5Q$, where $Q = \sum q_e$. For each one of these nine combinations, we have generated two instances, one with $q_e = d_e$ (for the PCPP-LC) and another with randomly generated demands q_e.
- 18 instances obtained from three Rural Postman Problem (RPP) instances proposed by Christofides et al. [8], P1, P2 and P4, with $|V| = 11, 14, 17$ and $|E| = 13, 32, 35$, respectively. The RPP differs from the CPP in the fact that only a subset of required

edges must be traversed. To obtain CPP-LC instances, we consider that all the edges in the graph must be traversed. For each graph, we generate six instances proceeding as above.

- 24 instances obtained from 12 RPP instances proposed by Hertz et al. [17]: r1 to r8 (with $|V| \in [6, 14]$ and $|E| \in [11, 48]$), d10, d11, g10 and g11 (with $|V| \in [18, 27]$ and $|E| \in [22, 33]$). For each graph, a proportional and a non-proportional instance was generated, all of them with $W = \frac{Q}{2}$.

All experiments were performed on an Intel Xeon E5-2650 v2 processor with a clock speed of 2.60 GHz and 64GB of RAM, running on Scientific Linux release 6.5 (Carbon). We used the MILP solver of CPLEX 12.6 with Concert Technology 2.9 with its default settings to solve the proposed two formulations. A time limit of one hour per instance was imposed.

Tables 1 to 3 present the results obtained with the arc routing formulation for both the proportional and non-proportional instances on the three sets. The first column is the name of the instance, while the second specifies whether the instance was solved to optimality (opt) or not (non-opt). The third column shows the percent gap between the upper and lower bound obtained and the fourth column provides the computing time in seconds. The last three columns present the same metrics for the proportional instances.

	Non-proportional q_e			Proportional q_e		
Instance		Gap (%)	Seconds		Gap (%)	Seconds
E7W0	opt	-	0.9	opt	-	36.0
E7WQ/2	opt	-	0.8	opt	-	6.9
E7W5Q	opt	-	0.4	opt	-	4.8
E10W0	opt	-	15.4	non-opt	3.7	3600
E10WQ/2	opt	-	10.5	opt	-	1633
E10W5Q	opt	-	7.6	opt	-	974.4
E20W0	non-opt	3.7	3600	non-opt	6.3	3600
E20WQ/2	opt	-	129.4	non-opt	2.8	3600
E20W5Q	opt	-	95.7	non-opt	1.0	3600

Table 1: Computational results on the Euclidean instances with the arc routing formulation

As can be seen, the CPP-LC is a very hard problem to solve. Even for very small instances CPLEX was not able to find an optimal solution within one hour of computing time. It is interesting to point out that the instances with demands proportional (equal) to the lengths were harder to solve than those with non-proportional demands. This is particularly surprising in the case of the Eulerian instances since they are polynomially solvable. For example, instance E10W0, associated with an Eulerian graph with 10 vertices and 18 edges and with demands equal to the lengths, ended with a gap of 5.9% after one hour. As expected, the larger the value of the curb weight W, the easier the instance. Recall that, as pointed out in Section 2, CPP-LC instances with very large values of W (compared with the demand values) have the same optimal tour as the classical CPP.

Since the results obtained with the arc routing formulation are far from being good, we have also run CPLEX MIP solver with the node routing formulation for the same instances.

	Non-proportional q_e			Proportional q_e		
Instance		Gap (%)	Seconds		Gap (%)	Seconds
P1W0	opt	-	10.3	opt	-	35.9
P1WQ/2	opt	-	2.7	opt	-	5.9
P1W5Q	opt	-	0.8	opt	-	1.6
P2W0	non-opt	24.3	3600	non-opt	24.9	3600
P2WQ/2	non-opt	8.5	3600	non-opt	10.7	3600
P2W5Q	non-opt	2.0	3600	non-opt	2.4	3600
P4W0	non-opt	23.5	3600	non-opt	32.6	3600
P4WQ/2	non-opt	10.1	3600	non-opt	21.8	3600
P4W5Q	non-opt	8.6	3600	non-opt	10.0	3600

Table 2: Computational results on the Christofides et al. [8] instances with the arc routing formulation

	Non-proportional q_e			Proportional q_e		
Instance		Gap (%)	Seconds		Gap (%)	Seconds
hertzr1	opt	-	0.6	opt	-	31.2
hertzr2	opt	-	9.7	opt	-	209.8
hertzr3	opt	-	54.9	non-opt	8.9	3600
hertzr4	non-opt	15.2	3600	non-opt	15.0	3600
hertzr5	opt	-	768.3	non-opt	3.6	3600
hertzr6	opt	-	1597.5	non-opt	12.9	3600
hertzr7	non-opt	12.3	3600	non-opt	21.5	3600
hertzr8	non-opt	4.4	3600	non-opt	10.8	3600
hertzd10	non-opt	26.4	3600	non-opt	25.3	3600
hertzd11	non-opt	22.3	3600	non-opt	22.9	3600
hertzg10	non-opt	2.6	3600	non-opt	6.0	3600
hertzg11	non-opt	13.8	3600	non-opt	15.6	3600

Table 3: Computational results on the Hertz et al. instances with the arc routing formulation

	F2			F1		
Instance	Optimal?	Gap (%)	Seconds	Optimal?	Gap (%)	Seconds
E7W0 (NP)	non-opt	5.5	3600	opt	-	0.9
E7W0 (P)	non-opt	23.4	3600	opt	-	36.0
E7WQ/2 (NP)	opt	-	91.2	opt	-	0.8
E7WQ/2 (P)	opt	-	537.1	opt	-	6.9
E7W5Q (NP)	opt	-	3.1	opt	-	0.4
E7W5Q (P)	opt	-	4.8	opt	-	4.8

Table 4: Computational results on the E7 instances

Tables 4 to 10 show the results obtained on these instances, where F1 corresponds to the arc routing formulation and F2 to the node routing one. Although formulation F2 seemed more promising because its original objective function is linear, the results obtained are even worse. In 13 of the instances for which F1 obtains the optimal solution, F2 fails to do so. Moreover, in the 13 instances in which both formulations reach the optimal solutions, the running time is noticeably larger with F2, while in those instances that are not solved by any of both formulations, the gaps obtained are much larger in the case of formulation F2. Note that F1 has around $6|E|^2$ variables and $10|E|^2$ constraints while F2 has approximately $16|E|^2$ variables and $4|E|^2$ constraints. As with the arc routing formulation, instances with proportional demands are harder than the non-proportional ones. Again, the difficulty of the instances decreases when the value for W increases.

	F2			F1		
Instance	Optimal?	Gap (%)	Seconds	Optimal?	Gap (%)	Seconds
E10W0 (NP)	non-opt	39.0	3600	opt	-	15.4
E10W0 (P)	non-opt	48.5	3600	non-opt	3.7	3600
E10WQ/2 (NP)	non-opt	12.4	3600	opt	-	10.5
E10WQ/2 (P)	non-opt	23.7	3600	opt	-	1633.0
E10W5Q (NP)	opt	-	792.3	opt	-	7.6
E10W5Q (P)	non-opt	3.5	3600	opt	-	974.4

Table 5: Computational results on the E10 instances

	F2			F1		
Instance	Optimal?	Gap (%)	Seconds	Optimal?	Gap (%)	Seconds
E20W0 (NP)	non-opt	65.8	3600	non-opt	3.7	3600
E20W0 (P)	non-opt	71.7	3600	non-opt	6.3	3600
E20WQ/2 (NP)	non-opt	32.3	3600	opt	-	129.4
E20WQ/2 (P)	non-opt	32.9	3600	non-opt	2.8	3600
E20W5Q (NP)	non-opt	9.8	3600	opt	-	95.7
E20W5Q (P)	non-opt	10.2	3600	non-opt	1.0	3600

Table 6: Computational results on the E20 instances

	F2			F1		
Instance	Optimal?	Gap (%)	Seconds	Optimal?	Gap (%)	Seconds
P1W0 (NP)	opt	-	3352.0	opt	-	10.3
P1W0 (P)	non-opt	16.6	3600	opt	-	35.9
P1WQ/2 (NP)	opt	-	62.7	opt	-	2.7
P1WQ/2 (P)	opt	-	112.8	opt	-	5.9
P1W5Q (NP)	opt	-	5.2	opt	-	0.8
P1W5Q (P)	opt	-	3.3	opt	-	1.6

Table 7: Computational results on the P01 instances

	F2			F1		
Instance	Optimal?	Gap (%)	Seconds	Optimal?	Gap (%)	Seconds
P2W0 (NP)	non-opt	75.8	3600	non-opt	24.3	3600
P2W0 (P)	non-opt	83.5	3600	non-opt	24.9	3600
P2WQ/2 (NP)	non-opt	35.5	3600	non-opt	8.5	3600
P2WQ/2 (P)	non-opt	41.9	3600	non-opt	10.7	3600
P2W5Q (NP)	non-opt	9.7	3600	non-opt	2.0	3600
P2W5Q (P)	non-opt	11.7	3600	non-opt	2.4	3600

Table 8: Computational results on the P02 instances

	F2			F1		
Instance	Optimal?	Gap (%)	Seconds	Optimal?	Gap (%)	Seconds
P4W0 (NP)	non-opt	77.7	3600	non-opt	23.5	3600
P4W0 (P)	non-opt	80.8	3600	non-opt	32.6	3600
P4WQ/2 (NP)	non-opt	39.1	3600	non-opt	10.1	3600
P4WQ/2 (P)	non-opt	42.9	3600	non-opt	21.8	3600
P4W5Q (NP)	non-opt	12.9	3600	non-opt	8.6	3600
P4W5Q (P)	non-opt	11.7	3600	non-opt	10.0	3600

Table 9: Computational results on the P04 instances

4 Two metaheuristics

Given that the sizes of the instances that can be optimally solved using the previous formulations is very small, we propose in this section two metaheuristics capable of producing good solutions on larger size instances. In particular, we have designed an Iterated Local Search (ILS) and a Variable Neighborhood Search (VNS). These metaheuristics exploit the fact that, given a sequence of edges without a specified direction of traversal, the cost of the best CPP-LC tour servicing the edges in that order can be computed in linear time.

4.1 A linear-time dynamic programming formulation to evaluate the cost of a CPP-LC tour

The minimal amount of information required to evaluate a CPP-LC tour is the sequence in which the edges should be serviced, and in which direction each edge should be traversed. A CPP-LC tour is then a vector of triplets $((i_1, j_1, d_1), (i_2, j_2, d_2), \ldots, (i_m, j_m, d_m))$, where the first two components of every triplet denote the edge being serviced, and the third component denotes the direction of traversal of an edge, with $d = 1$ implying from i to j, and $d = 2$ from j to i.

Following the example in Section 2, the tour $(1,2), [2,1], (1,3), [3,1], [1,2], (2,4), (4,3), [3,1]$ could be represented as $((1,2,1), (1,3,1), (2,4,1), (3,4,2))$, where the deadheads are implied by the mismatching endnodes of traversals.

	F2			F1		
Instance	Optimal?	Gap (%)	Seconds	Optimal?	Gap (%)	Seconds
hertzr1 (NP)	opt	-	11.3	opt	-	0.6
hertzr1 (P)	opt	-	492.7	opt	-	31.2
hertzr2 (NP)	opt	-	825.6	opt	-	9.7
hertzr2 (P)	non-opt	8.4	3600	opt	-	209.8
hertzr3 (NP)	non-opt	6.7	3600	opt	-	54.9
hertzr3 (P)	non-opt	15.9	3600	non-opt	8.9	3600
hertzr4 (NP)	non-opt	45.5	3600	non-opt	15.2	3600
hertzr4 (P)	non-opt	45.2	3600	non-opt	15.0	3600
hertzr5 (NP)	non-opt	41.1	3600	opt	-	768.3
hertzr5 (P)	non-opt	40.1	3600	non-opt	3.6	3600
hertzr6 (NP)	non-opt	24.2	3600	opt	-	1597.5
hertzr6 (P)	non-opt	35.0	3600	non-opt	12.9	3600
hertzr7 (NP)	non-opt	40.2	3600	non-opt	12.3	3600
hertzr7 (P)	non-opt	44.5	3600	non-opt	21.5	3600
hertzr8 (NP)	non-opt	34.3	3600	non-opt	4.4	3600
hertzr8 (P)	non-opt	38.3	3600	non-opt	10.8	3600
hertzd10 (NP)	non-opt	43.6	3600	non-opt	26.4	3600
hertzd10 (P)	non-opt	48.6	3600	non-opt	25.3	3600
hertzd11 (NP)	non-opt	46.9	3600	non-opt	22.3	3600
hertzd11 (P)	non-opt	50.1	3600	non-opt	22.9	3600
hertzg10 (NP)	non-opt	31.2	3600	non-opt	2.6	3600
hertzg10 (P)	non-opt	35.3	3600	non-opt	6.0	3600
hertzg11 (NP)	non-opt	36.7	3600	non-opt	13.8	3600
hertzg11 (P)	non-opt	43.4	3600	non-opt	15.6	3600

Table 10: Computational results on the Hertz et al. instances

With this representation, if the deadheading distances are precomputed, the cost of a given tour can be computed in $O(m)$ time. If we represent a solution only by the sequence of the edges to be serviced, there are 2^m associated CPP-LC tours depending on the direction in which each edge is traversed. We now show that it is still possible to compute the cost of the best of these tours in $O(m)$ time.

Theorem 5 *Given the sequence of edges to be serviced in a CPP-LC tour, the optimal direction of traversal for each edge and the resulting objective value can be computed in $O(m)$ time.*

Proof: We assume that the distances between all vertices of the graph are precomputed, which can be done in $O(n^3)$ time using the Floyd-Warshall algorithm [16] for all-pairs shortest paths. We also precompute the remaining amount of demand on board just before servicing the k^{th} edge in the sequence and we denote it as Q_k. This computation can also be implemented to run in $O(m)$ time using a suffix sum. Let $f_k(d)$ denote the minimum cost of completing the partial tour that starts from the k^{th} edge in the sequence, when the last traversal has been in direction d. The dynamic programming recursion is then

$$f_1(1) = \big(W + Q - \frac{q_1}{2}\big)d_{i_1,j_1} + \min\left\{\begin{array}{l}(W+Q)d_{1,i_1} + f_2(1)\\ (W+Q)d_{1,j_1} + f_2(2)\end{array}\right. \tag{25}$$

$$f_k(1) = \big(W + Q_k - \frac{q_k}{2}\big)d_{i_k,j_k} + \min\left\{\begin{array}{l}(W+Q_k)d_{j_{k-1},i_k} + f_{k+1}(1)\\ (W+Q_k)d_{j_{k-1},j_k} + f_{k+1}(2)\end{array}\right., \; k = 2,\ldots,m-1 \tag{26}$$

$$f_k(2) = \big(W + Q_k - \frac{q_k}{2}\big)d_{i_k,j_k} + \min\left\{\begin{array}{l}(W+Q_k)d_{i_{k-1},i_k} + f_{k+1}(1)\\ (W+Q_k)d_{i_{k-1},j_k} + f_{k+1}(2)\end{array}\right., \; k = 2,\ldots,m-1 \tag{27}$$

$$f_m(1) = \big(W + \frac{q_m}{2}\big)d_{i_m,j_m} + \min\left\{\begin{array}{l}(W+q_m)d_{j_{m-1},i_m} + Wd_{j_m,1}\\ (W+q_m)d_{j_{m-1},j_m} + Wd_{i_m,1}\end{array}\right. \tag{28}$$

$$f_m(2) = \big(W + \frac{q_m}{2}\big)d_{i_m,j_m} + \min\left\{\begin{array}{l}(W+q_m)d_{i_{m-1},i_m} + Wd_{j_m,1}\\ (W+q_m)d_{i_{m-1},j_m} + Wd_{i_m,1}.\end{array}\right. \tag{29}$$

The first term on the right-hand side of every equation is the traversal cost of the corresponding edge. The first term within each 'min' operator for $k = 1,\ldots,m-1$ represents the deadheading cost, which may be zero if the endpoint of the last traversal and the starting point of the current traversal coincide. For $k = 1$, the deadheading distance for the first term is computed from the depot. The last term, for $k = 1,\ldots,m-1$, corresponds to the cost of the later stages. When $k = m$, the last term is the cost of returning to the depot with the curb weight.

There are $2m$ stages, the complexity of each one being $O(1)$, the overall time complexity is therefore $O(m)$. Clearly, the space complexity is also $O(m)$. ♦

The importance of this result can be better appreciated by considering that the total number of possible tours is 2^m and the complexity of each iteration of a local search algorithm for the traversal directions is also $O(m)$. As a result, we can use the simplified solution representation consisting of just the sequence of edges to be serviced and compute the traversal directions as required.

Now that the solution representation is simplified to a sequence, we can use the three operators 1-OPT, 2-OPT, and 2-EXCHANGE. The 1-OPT operator consists of finding one edge to relocate from its position within the sequence to a better one; the 2-OPT operator corresponds to finding a subsequence of edges to reverse; the 2-EXCHANGE operator consists of finding two edges within the sequence to swap their positions. All of them run on a best-improvement strategy.

4.2 Pseudo-codes of the metaheuristics

Based on the theorem regarding the use of dynamic programming to compute the optimum cost of a given sequence in $O(m)$ time, we have opted to represent the solutions of CPP-LC as sequences of edges. For the algorithms below, we denote sequences of edges as σ, and the DP result for a given sequence σ as $z(\sigma)$.

```
Sort the edges in decreasing order of demand multiplied by distance;
Initialize σ* as an empty sequence;
for i = 1 to m do
    z_min = ∞;
    for j = 1 to i do
        Create σ' by inserting the i^th edge in the list in the j^th position of σ*;
        if z(σ') < z_min then
            z_min = z(σ');
            σ'' = σ';
        end
    end
    σ* = σ'';
end
return σ*
```

Algorithm 1: Greedy Constructive heuristic

```
Call Greedy Constructive to obtain σ*;
σ' = σ*;
for k = 1 to k_max do
    Perturb σ' by performing m/5 random exchanges of edges within the sequence;
    Apply local search on σ' by selecting the best move among the operators 1-OPT,
    2-OPT, and 2-EXCHANGE;
    if z(σ') < z(σ*) then
        σ* = σ';
    else
        σ' = σ*;
    end
end
return σ*
```

Algorithm 2: Iterated Local Search metaheuristic

```
Call Greedy Constructive to obtain σ*;
σ' = σ*;
for k = 1 to k_max do
    Perturb σ' by performing m/5 random exchanges of edges within the sequence;
    Apply local search on σ', using the operators 2-EXCHANGE, 1-OPT, 2-OPT in
    the given order, stopping and applying the first improving operator in every step of
    the local search;
    if z(σ') < z(σ*) then
        σ* = σ';
    else
        σ' = σ*;
    end
end
return σ*
```

Algorithm 3: Variable Neighbourhood Search metaheuristic

Both ILS and VNS are based on the principle of perturbing the best known solution for *diversification* and applying local search for *intensification*. The main difference is the way in which the local search operators are handled. In ILS, all operators are considered simultaneously, and the best move is selected. In VNS, the operators are tried in a given order (in some implementations, on a random order), and the first operator that returns an improvement is applied. In our implementation of the VNS, we have used the order 2-EXCHANGE, 1-OPT, 2-OPT, i.e. in increasing order of computational cost.

4.3 Computational results for the metaheuristics

We have implemented the metaheuristics in C++, and performed the computational experiments on the machine described in Section 3.3.

In order to check the quality of the bounds provided by the metaheuristics, we have first tested the ILS and VNS on the instances described in Section 3.3, using $k_{max} = 300$. Each instance was solved only once for all the 26 instances for which the exact algorithm obtains the optimal solution, the heuristic procedures also find it. The results are reported in Table 11. The results for the 34 unsolved instances are summarized in Table 12. The first column shows the name of the instance set, and the number of unsolved instances is given in the second column. Columns 3 and 4 show the average lower and upper bounds obtained with the exact algorithm for formulation F1, while columns 5 and 6 report the average of the best bound obtained with the two heuristics and the percent gap of the average upper bound with respect to the average lower bound. Finally, the last column shows the average running time of the heuristic algorithms in seconds.

In all the instances, the upper bound obtained with the heuristic algorithms is better than the upper bound found by the exact method after one hour of computing time. The gaps obtained are also very good (except for instances in sets Hr and Hd) considering the lower bounds may be still far from the optimal value.

To test the behavior of the proposed metaheuristics on larger instances, we have generated two new sets of instances as follows. Instance generation is based on the number of vertices,

Set	# opt	Optimal value	Heur UB	Seconds Heur	Seconds Exact
E07	6/6	9640.3	9640.3	0.05	8.3
E10	5/6	269193.5	269193.5	0.27	528.2
E20	2/6	686418.0	686418.0	2.90	112.6
P01	6/6	12964.4	12964.4	0.08	9.5
Hr	7/16	388576.7	388576.7	0.32	381.7

Table 11: Comparison of the exact and heuristic algorithms on the optimally solved instances

Set	# non-opt	F1 LB	F1 UB	Heur UB	Gap (%)	Seconds
E10	1/6	80236.3	85284.5	85284.5	5.92	0.4
E20	4/6	1198075.8	1237876.4	1226363.9	2.31	4.3
P02	6/6	452900.4	476726.7	470545.4	3.75	6.1
P04	6/6	61964.6	68380.3	66912.3	7.38	8.2
Hr	9/16	1732941.0	2094019.8	1931421.1	10.28	10.0
Hd	4/4	209610.6	608530.9	244967.1	14.43	46.7
Hg	4/4	2466.6	2882.0	2623.3	5.97	4.2

Table 12: Comparison of the exact and heuristic algorithms on the unsolved instances

n, and three parameters: the sparsity parameter α, the correlation parameter κ, and the curb weight ratio parameter ρ. Instances were generated by first generating n random points in the plane with both coordinates in the interval [0, 100]. The distances between the vertices were computed using the Euclidean distances and rounding them up. Next, the minimum spanning tree was determined using Prim's algorithm [29], and the edges in the tree were added to the edge list to ensure connectivity. In addition, $\lfloor \alpha(n(n-1)/2-(n-1)) \rfloor$ shortest unused edges were added to the edge list, where higher values of α result in a higher number of edges. The demand of each selected edge (i, j) was determined as $q_{ij} = \lceil \kappa d_{ij} + (1-\kappa)U[0,1]d_{ij} \rceil$, which results in demand values with a higher correlation to d_{ij} as κ increases. Finally, the curb weight of the vehicle was determined as $W = \rho \sum_{(i,j)\in E} q_{ij}$.

For a comprehensive testing of the algorithm performance, we have generated 10 instances for $n \in \{10, 20\}$ and each setting of $(\alpha, \kappa, \rho) \in \{0.2, 0.5, 0.8\}^3$, resulting in 540 instances. A preliminary analysis of the results has shown that as the sparsity parameter increases above 0.5, the resulting routes contains very few deadheadings, so the resulting instances are not very challenging. The effects of the other two parameters are observed to be negligible. As a result, we have also created a second set of more difficult instances by generating 10 instances for $n \in \{10, 20, 30\}$ and each setting of $(\alpha, \kappa, \rho) \in \{0.2, 0.33, 0.5\}^3$ for a total of 810 instances.

The aggregate computational results for the first and second instance sets are presented in Tables 13 and 14, respectively. In these tables, the columns labeled $|V|$ and $|E|$ represent the number of vertices and edges of the 90 instances in the corresponding row. The columns labeled 'EC deviation', 'ILS deviation' and 'VNS deviation' show the percent average gap obtained with the greedy constructive, the ILS and the VNS with respect to the best known solution. The average computing times in seconds are provided in columns 'ILS seconds' and 'VNS seconds'.

$\|V\|$	$\|E\|$	EC deviation	ILS deviation	ILS seconds	VNS deviation	VNS seconds
10	16	111.65%	0.01%	0.19	0.00%	0.10
10	27	76.43%	0.00%	2.04	0.01%	1.15
10	37	60.80%	0.00%	9.34	0.01%	5.95
20	53	152.43%	0.01%	42.30	0.04%	27.98
20	104	90.71%	0.01%	858.29	0.03%	674.35
20	155	68.31%	0.01%	5202.90	0.01%	4450.49

Table 13: Computational results with $k_{max} = 300$

$\|V\|$	$\|E\|$	EC deviation	ILS deviation	ILS seconds	VNS deviation	VNS seconds
10	16	110.96%	0.04%	0.04	0.01%	0.02
10	20	98.93%	0.04%	0.13	0.01%	0.06
10	27	73.90%	0.04%	0.46	0.01%	0.24
20	53	156.70%	0.12%	6.89	0.05%	3.93
20	75	120.43%	0.05%	40.23	0.05%	26.68
20	104	92.76%	0.03%	185.85	0.03%	135.67
30	110	171.87%	0.08%	187.37	0.08%	131.11
30	162	127.82%	0.04%	1200.48	0.03%	957.01
30	232	98.05%	0.02%	6714.52	0.02%	5752.62

Table 14: Computational results with $k_{max} = 75$

For the first set of instances, we have used $k_{max} = 300$, an iteration limit that would keep the CPU time requirement at reasonable levels for large values of $m = |E|$. Both metaheuristics are comparable, ILS outperforming VNS for larger instances, and VNS requiring a shorter CPU time. The performance of the greedy heuristic improves as m increases. For the second instance set, we had to use $k_{max} = 75$ to be consistent among the instances and still have a reasonable CPU time for the largest instances. In this case, VNS slightly outperforms ILS due to its faster convergence rate. The observations about the CPU time requirement of the metaheuristics and the performance of the greedy algorithm remain unchanged.

We have run the exact algorithm on the 270 smallest instances of the first set, those with $|V| = 10$ and $|E| \in \{16, 27, 37\}$. The exact method was capable of solving to optimality all the 90 instances, except one with $|E| = 16$, within an average computing time of 226 seconds. For these 89 instances, VNS also provided an optimal solution with an average computing time of 0.1 second, while ILS took 0.2 second on average and found 87 optima. On the other two instances the deviation gaps from the optimal values were 0.12% and 0.88%. Table 15 shows the average lower and upper bounds obtained with the exact algorithm and the heuristic procedures on the 180 remaining instances. The feasible solution values provided by ILS and VNS are considerably better than the upper bounds obtained by the exact method after one hour of computing time, and show a gap of approximately 5% with respect to the lower bound. These results confirm that both heuristic procedures are capable of providing solutions of good quality within very short computing times.

	# inst	F1		ILS		VNS	
		LB	UB	UB	Seconds	UB	Seconds
$\|E\| = 27$	90	763864.9	809948.8	803533.8	1.9	803557.7	1.1
$\|E\| = 37$	90	1986997.2	2195655.6	2102465.5	8.9	2102584.0	5.7

Table 15: Comparison of the exact and heuristic algorithms

5 Conclusions

We have presented a new and interesting arc routing problem in which the cost of traversing the edges depends not only on their length, but also on the load of the vehicle when traversing them. This new problem, called the Chinese Postman Problem with Load-Dependent Costs, has been proved to be strongly NP-hard, although it is solvable in polynomial time in some special cases, namely on star trees, and when the demands of the edges are proportional to their lengths and the graph is Eulerian or a weighted tree. For the general problem, we have proposed two formulations: a pure arc routing formulation and a node routing formulation based on a transformation of the original graph. Computational experiments with both formulations have shown that they can only solve very small instances optimally within a reasonable time. Given the difficulty of the problem, we have proposed two metaheuristics based on the ILS and VNS methodologies. The results obtained with both methods on a very large set of instances demonstrate that these algorithms provide very good feasible solutions within short computing times.

Acknowledgments: The work by Ángel Corberán, Isaac Plana, and José M. Sanchis was supported by the Spanish Ministerio de Economía y Competitividad and Fondo Europeo de Desarrollo Regional (FEDER) through project MTM2015-68097-P (MINECO/FEDER) and by the Generalitat Valenciana (project GVPROMETEO2013-049). Gilbert Laporte was supported by the Canadian Natural Sciences and Engineering Research Council under grant 2015-06189. Thanks are due to the referees for their valuable comments.

References

[1] F. Afrati, S. Cosmadakis, C. Papadimitriou and N. Papakostantinou (1986). The complexity of the traveling repairman problem. *RAIRO Informatique Théorique et Applications* 20, 79-87.

[2] M. Barth and K. Boriboonsomsin (2008), Real-world impacts of traffic congestion. *Transportation Research Record: Journal of the Transportation Research Board* 2058, 163-171.

[3] M. Barth, T. Younglove and G. Scora (2005), Development of a heavy-duty diesel modal emissions fuel consumption model. Technical Report, UC Berkeley: California Partners for Advanced Transit and Highways (PATH).

[4] T. Bektaş, E. Demir and G. Laporte (2016), Green vehicle routing. In: Psaraftis, H.N. (Ed.), *Green Transportation Logistics: The Quest for Win-Win Solutions*. Springer, Cham, Switzerland, pp. 243-265.

[5] T. Bektaş and G. Laporte (2011), The Pollution-Routing Problem. *Transportation Research Part B* 45, 1232-1250.

[6] H. Belouadah, M.E. Posner and C.N. Potts (1992), Scheduling with release dates on a single machine to minimize total weighted completion time. *Discrete Applied Mathematics* 36, 213-231.

[7] A. Blum, P. Chalasani, D. Coppersmith, B. Pulleyblank, P. Raghavan and M. Sudan (1994), The minimum latency problem. In *Proc. 26th ACM Symposium on the Theory of Computing*, 163-171.

[8] N. Christofides, V. Campos, Á. Corberán and E. Mota (1981), An algorithm for the Rural Postman Problem. *Technical Report ICOR 81.5*, Imperial College, London.

[9] Á. Corberán and G. Laporte (2014), Arc Routing: Problems, Methods, and Applications. MOS-SIAM Series on Optimization 20, SIAM, Philadelphia.

[10] S. Dabia, E. Demir and T. Van Woensel (2016), An exact approach for a variant of the pollution-routing problem. *Transportation Science*, forthcoming.

[11] E. Demir, T. Bektaş and G. Laporte (2011), A comparative analysis of several vehicle emission models for freight transportation. *Transportation Research Part D* 16, 347-357.

[12] E. Demir, T. Bektaş and G. Laporte (2012), An adaptive large neighborhood search heuristic for the pollution-routing problem. *European Journal of Operational Research* 223, 346-359.

[13] E. Demir, T. Bektaş and G. Laporte (2014a), The bi-objective pollution-routing problem. *European Journal of Operational Research* 232, 464-478.

[14] E. Demir, T. Bektaş and G. Laporte (2014b), A review of recent research on green road freight transportation. *European Journal of Operational Research* 237, 775-793.

[15] K. Fagerholt, G. Laporte and I. Norstad (2010), Reducing fuel emissions by optimizing speed on shipping routes. *Journal of the Operational Research Society* 61, 523-529.

[16] R.W. Floyd (1962), Algorithm 97 - Shortest path. *Communications of the ACM* 5, 345.

[17] A. Hertz, G. Laporte and P. Nanchen-Hugo (1999), Improvement procedures for the Undirected Rural Postman Problem. *INFORMS Journal on Computing* 11, 53-62.

[18] L.M. Hvattum, I. Norstad, K. Fagerholt, and G. Laporte (2013), Analysis of an exact algorithm for the vessel speed optimization problem. *Networks* 62, 132-135.

[19] I. Kara, B.Y. Kara and M.K. Yetis (2007), Energy minimizing vehicle routing problem. In: Dress, A., Xu, Y., Zhu, B. (Eds.), *Combinatorial Optimization and Applications*. Lecture Notes in Computer Science 4616, Springer, Berlin/Heidelberg, pp. 62-71.

[20] Ç. Koç, T. Bektaş, O. Jabali and G. Laporte (2014), The fleet size and mix pollution-routing problem. *Transportation Research Part B* 70, 239-254.

[21] Ç. Koç, T. Bektaş, O. Jabali and G. Laporte (2016a), The impact of location, fleet composition and routing on emissions in urban logistics. *Transportation Research Part B* 84, 81-102.

[22] Ç. Koç, T. Bektaş, O. Jabali and G. Laporte (2016b), A comparison of three idling options in long-haul truck scheduling, *Transportation Research Part B* 93, 631-647.

[23] R. Kramer, A. Subramanian and T. Vidal (2015), A matheuristic approach for the pollution-routing problem. *European Journal of Operational Research* 243, 523-539.

[24] G. Laporte (2014), The undirected Chinese postman problem. In: Corberán, Á., Laporte, G. (Eds.), *Arc Routing: Problems, Methods, and Applications.* MOS-SIAM Series on Optimization 20, SIAM, Philadelphia, pp. 53-64.

[25] A. McKinnon (2007), CO_2 Emissions from freight transport in the UK. TR, Prepared for the *Climate Change Working Group of the Commission for Integrated Transport, London, UK.* `<www.isotrak.com/news/press/CO2_emissions_freight_transport.pdf>` (accessed 11.02.11).

[26] I. Norstad, K. Fagerholt and G. Laporte (2011), Tramp ship routing and scheduling with speed optimization. *Transportation Research Part C* 19, 853-865.

[27] N. van Omme (2011), Le problème du postier chinois cumulatif. PhD thesis, *Université de Montréal.*

[28] N. van Omme, M. Gendreau and P. Soriano (2013), On the complexity of the Cumulative Chinese Postman Problem. *Technical Report*, CIRRELT, Montréal.

[29] R.C. Prim (1957), Shortest connection networks and some generalizations. *Bell System Technical Journal* 36, 1389-1401.

[30] M. Ross (1994), Automobile fuel consumption and emissions: effects of vehicle and driving characteristics. *Annual Review of Energy and the Environment* 19, 75-112.

[31] A. Sbihi and R.W. Eglese (2007), Combinatorial optimization and green logistics. *4OR: A Quarterly Journal of Operations Research* 5, 99-116.

[32] R. Sitters (2002), The minimum latency problem is NP-hard for weighted trees. In: *Proc. 9th International IPCO Conference on Integer Programming and Combinatorial Optimization*, 230-239.

[33] E.E. Zachariadis, C.D. Tarantilis and C.T. Kiranoudis (2015), The load-dependent vehicle routing problem and its pick-up and delivery extension. *Transportation Research Part B* 71, 158-181.